PoolBot: Physics-Aware Mobile Manipulator for Multi-Shot Autonomous Billiards

Avi Narula
Electrical Engineering with Computing
Massachusetts Institute of Technology
Cambridge, United States
avin@mit.edu

Leo Sun
Artificial Intelligence and Decision Making
Massachusetts Institute of Technology
Cambridge, United States
leosun@mit.edu

Suraj Reddy
Electrical Engineering with Computing
Massachusetts Institute of Technology
Cambridge, United States
surajrdy@mit.edu

***Abstract*—This paper presents PoolBot, a physics-aware mobile manipulation system for autonomous pool playing on a standard pool table. PoolBot moves beyond prior fixed-base and single-shot systems by integrating whole-table mobility and physics-based shot planning to play multi-shot sequences and successfully score more than 1 ball through our full-stack system. Our system combines a pool planning pipeline, a mobile base that repositions around the table to maintain feasible shot geometry, and a manipulator that executes strikes with an inverse dynamics controller generated by a trajectory-checked shot planner. The planner evaluates collisions, feasibility, and pocketing likelihood to select optimal shots in sequence. In simulation, PoolBot achieves 55% shot success rate and clears an average of 2.6 balls per run. These results demonstrate that mobile manipulation coupled with physics-aware pool solving enables robust multi-shot autonomous billiards.**

I. Introduction

The game of pool involves using a cue/stick to strike the cue ball, aiming to hit the other balls into the pockets at the edges of the table. The game's complexities, which make pool even difficult for humans, come from planning the optimal shot to hit and accurately hitting the ball [1]. Our idea is inspired by the Outstanding Project "An Autonomous Pool-Playing Robot" by Jinger Chong and Eugenia Feng, which had a few limitations as outlined in their video: (1) Only pocketed a single ball. (2) The robotic arm was fixed to one side of the table. We plan to solve these limitations and are aiming to have a mobile manipulator with planning that maximizes the number of balls successfully pocketed given a random game state with only one type of balls. We originally aimed to pocket a full rack of 15 balls, but to keep our project within scope, we decided to focus on scoring more than one ball in a single run.

From a robotics perspective, pool provides a compact benchmark for several core manipulation challenges: (1) high-precision contact initiation, where millimeter-scale deviations produce large outcome differences; (2) physics-aware planning, requiring prediction of ball trajectories, rebounds, and pocket entry margins; and (3) mobile manipulation, where the robot must reposition itself to maintain feasible strike geometry. These challenges make pool an effective testbed for studying how robots integrate physical dynamics and motion planning in a closed-loop system. By solving pool, we effectively solve a simplified version of real-world tasks like clearing debris, constrained reaching, and long-horizon manipulation planning.

This project is interesting to us because it combines something we genuinely enjoy, playing pool, with the technical challenges of robotics. Ultimately, this pool "game" serves as a proxy for solving greater engineering challenges, such as physics-based intermediate-horizon decision making, along with transferring logic solvers to the physical world. Possible applications of the work we will do will relate to things like disaster recovery, for example, where robots must understand the consequences of their actions in the physical world (i.e., moving rubble out of the way without injuring a human). Additionally, another interesting aspect is how we combine an arm manipulator with a mobile base, creating a system that can move around the table, line up shots, and shoot balls in a coordinated way, which would be quite similar to mobile manipulators we deploy in the real world!

II. Related Works

As mentioned above, our project is building off of one of the previous 6.4210/6.4212 Outstanding Projects, "An Autonomous Pool-Playing Robot" by Jinger Chong and Eugenia Feng [4]. This set the foundations of our problem; however, we are planning to address its shortcomings, namely, Mobile Manipulation and Long-Horizon Gameplay. Chong and Feng's work was motivated by combining multiple methodologies in class to create a proof of concept for manipulator capable of hitting a ball, but was mainly limited as they could only hit one ball. In particular, shot selection was governed by a simple angular heuristic and did not consider positional play, a key tactic in long-horizon pool games. We retain the same basic cue-ball to object-ball to pocket formulation but extend this line of work along two main points: (i) enabling multi-shot sequences that explicitly reason about the changing table configuration and (ii) introducing full-table mobility so the robot can reposition in accordance to position play.

More broadly, prior work on pool-playing robots explores complementary approaches to perception, planning, and control. Greenspan's "Toward a Competitive Pool-Playing Robot" [1] analyzes the geometry and physics of billiards, emphasizing accurate modeling of ball trajectories, cushions, and

pocket entry tolerances. This drives our motivation to use a physics-aware shot planner that explicitly checks for collisions and pocketability rather than relying solely on local heuristics. Other projects such as Yale's "Creating an Autonomous System to Play Pool" [2] and Stanford's "Deep Cue Learning" [3] combine vision-based game state identification with learning-based or reinforcement-learning (RL) strategies for shot selection. These studies demonstrate the upper bound of how RL policies can achieve strong performance given extensive simulated experience, but they did not deal with mobile manipulators, and require substantial training time and data.

From our investigation of these studies, we conclude that heavy-weight deep perception and RL policies are not strictly necessary for our setting, where the geometry is structured and the table dynamics are relatively well understood. Instead, we have chosen to adopt a Kinematics/Dynamics-focused stack tailored to our system modules. For planning, we build on the geometric insights of [1] and [4] to design a sampling-based, physics-consistent shot planner that can be evaluated without full dynamic simulation. Finally, unlike the non-mobile manipulator systems in [2]–[4], we pair this planner with a mobile manipulator capable of moving around the table, sequencing many shots, and exploiting full-table mobility to recover from otherwise infeasible configurations.

III. Approach

A. System Overview

Given a current table configuration, PoolBot operates in a closed-loop state machine that integrates planning, control, and mobility (Fig 1). Firstly, we pass ball positions obtained from the plant to the physics-aware shot planner (Sec C), which samples candidate cue directions, checks for collisions, and selects a ball–pocket–cue direction combination with high pocketing likelihood. The selected shot defines a desired cue-ball contact point and cue angle.

The full-table mobility and shot-execution module (Sec. D and E) then plans a collision-free path for the mobile base around the table to reach a pose that can successfully solve for the chosen cue direction, and generates a striking trajectory for the arm that aligns the cue stick with the cue ball and executes the shot. After the shot is simulated in Drake, the resulting ball configuration is fed back into the planning pipeline, and the loop repeats until no feasible shots remain.

Inverse kinematic failures are handled in the state machine by trying the next best shot instead. The state machine also ends if there are three inverse kinematic failures in a row or if there are no more possible shots to try.

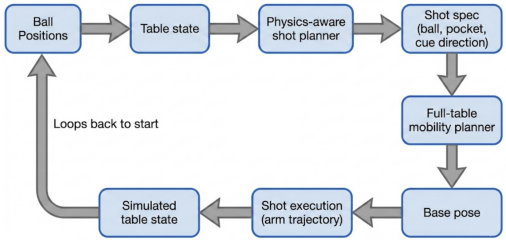

Fig. 1. State Machine representation of closed-loop process of PoolBot

We implement and evaluate PoolBot in the Drake physics simulator using custom pool table, ball, and cue models defined in SDF. The table geometry is loaded from `pool_table.sdf`, which specifies an 8-foot style playing surface approximated as a rectangular box of size $2.240 \times 1.120 \times 0.050$ m. Additional box-shaped collision geometries model the side rails and the corner and middle pocket bevels, while the pocket openings are represented by gaps in the rail collision geometry so that balls falling through these regions below a certain z threshold are considered pocketed.

Pool balls are modeled as rigid spheres using the `primitive_blueball.sdf` model. Each ball has mass $m = 0.17$ kg and radius $R = 0.028575$ m (a regulation 2.25-inch billiard ball), with diagonal inertia entries of 5.6×10^{-5} kg $\cdot$ m^2. The cue stick is represented by the `cue.sdf` model as a cylinder of radius 0.01 m and length 0.30 m with mass 0.5 kg. Gravity is set to 9.81 m/s^2 in the world frame.

The robot model follows the Mobile IIWA setup used in the course examples with the caveat of mobility: a KUKA iiwa arm mounted on a planar mobile base that can translate and rotate around the table. The environment is constructed by a helper routine (`setup_pool_environment` in our code) that instantiates the robot, table, balls, and cue in a single `MultibodyPlant`, wires the controllers, and attaches a MeshCat visualizer for rendering (Fig 2). Contact between balls, cue, and table is handled by Drake's compliant contact and Coulomb friction models. We do not explicitly model spin-dependent effects or cloth deformation; instead, the simulation captures the dominant rigid-body interactions (ball–ball, ball–rail, and cue–ball impacts) while remaining fast enough to roll out many shots.

Unless otherwise specified in the evaluation section, we simulate one cue ball and a set of object balls placed on the table by a random sampler that enforces non-overlap and keeps balls within the playable region. We have access to "cheat ports" that give the ground-truth ball poses from the simulator.

B. Simulation Details

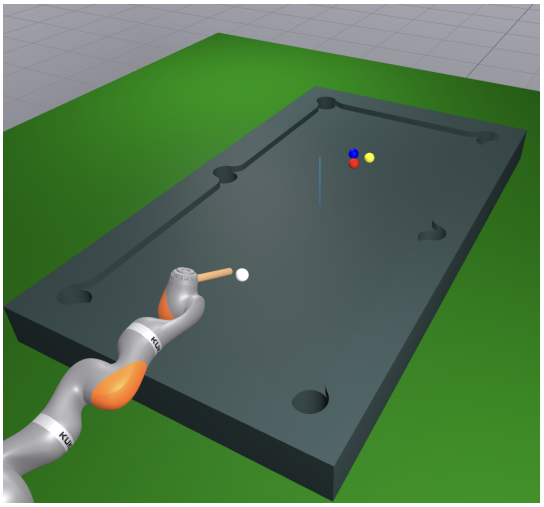

Fig. 2. The setup of the simulation is pictured. Utilizing Inverse Kinematics to select a cue stick position that targets the cue ball at the desired angle.

C. Friction

While Drake's hydro-elastic collisions have many benefits, the hydro-elastic friction was weak despite high friction coefficients, leading to balls never slowing down. We hypothesize this was the case for two reasons: (1) There is little contact area between a ball and table as well between two balls. (2) The balls after being hit end up rolling with slipping which is physically hard to model.

Our solution was to add another controller to our simulation diagram which resisted the motion with a proportional spatial force to spatial velocity. We tune the proportionality constant to 0.08 which result in visually reasonable friction interactions. Additionally, as soon as the ball was below $5e-2$, we increase the proportionality constant to immediately decrease the velocity to 0. This ensured that each strike occurred with a fixed table state.

$$\begin{bmatrix} \boldsymbol{\tau} \\ \mathbf{f} \end{bmatrix} = -c \begin{bmatrix} \gamma \mathbf{I}_3 & \mathbf{0} \\ \mathbf{0} & \mathbf{P}_{xy} \end{bmatrix} \begin{bmatrix} \boldsymbol{\omega} \\ \mathbf{v} \end{bmatrix}$$

where c is the damping coefficient, $\gamma = 0.05$ is a rotational scaling factor, and $\mathbf{P}_{xy} = \mathrm{diag}(1, 1, 0)$ projects the velocity onto the table plane.

D. Physics-aware Shot Planner

The shot selection module identifies a feasible and high-probability pocketing shot by simulating cue-ball interactions over a dense set of candidate orientations. Rather than solving a continuous nonlinear program directly, we adopt a sampling-based optimization strategy that evaluates thousands of potential cue stick angles and selects the angle–ball–pocket combination with the highest empirical success rate. This algorithm is visualized in Fig 3.

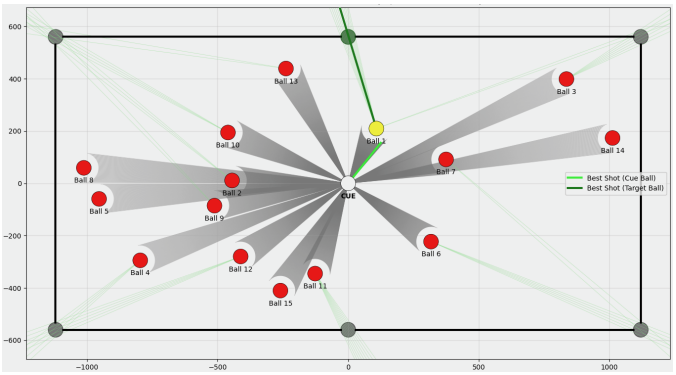

Fig. 3. Figure showcases a shot selection planning algorithm that optimizes based on the projected cue ball trajectory and the remaining balls on the board

We parameterize each candidate shot by an angle $\theta \in [0, 2\pi)$, which induces a cue-ball direction

$$d_c(\theta) = \begin{bmatrix} \cos\theta \\ \sin\theta \end{bmatrix},$$

and a corresponding ray

$$\ell_c(\theta) = \{x_c + t\, d_c(\theta) : t \geq 0\},$$

where x_c is the cue ball position. For each sampled angle, we check whether this ray intersects a target ball x_i at the correct collision radius $2R$ by solving the quadratic line–circle intersection equation

$$\|x_c + t\, d_c(\theta) - x_i\|^2 = (2R)^2,$$

and require a feasible root $t > 0$. If an intersection exists, the solver verifies that the path to x_i is unoccluded: any secondary ball x_j satisfying

$$\mathrm{dist}(x_j,\, \ell_c(\theta)) \leq 2R$$

invalidates the shot.

Upon a valid collision, the outgoing direction of the target ball is approximated using a point-contact model, and we check whether the resulting trajectory reaches a legal pocket p_k by solving

$$\|x_i + t\, d_i - p_k\|^2 = R_p^2,$$

while also ensuring that no intervening balls intersect this path.

Each target ball and pocket pair is scored by counting the number of sampled angles θ_m that result in a successful pocketing shot. This defines a robustness score

$$R_{i,k} = \sum_m \mathbf{1}\{\theta_m \text{ yields a valid hit of ball } i \text{ into pocket } k\},$$

where the indicator is 1 if the angle results in a feasible pocketing shot and 0 otherwise. The best shot is then defined as the (i, k) pair with the largest robustness score, and the median successful angle is selected as the execution angle for downstream motion planning. This approach yields a practical, physics-consistent shot planner that handles ball occlusions, feasibility constraints, and pocket geometry without requiring full dynamic simulation.

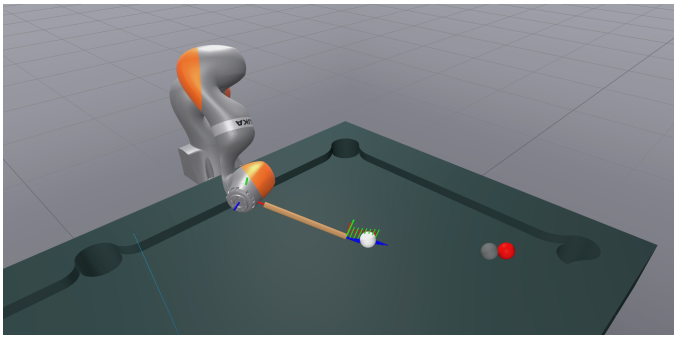

Fig. 4. The trajectory key frames are visualized along with the desired collision point (in gray) of the cue ball

E. Full-table Mobility

The motion planning around the pool table is broken up into two heuristics for (1) determining where the base should be based on the selected shot and (2) moving the base around the table while avoiding collisions. Both heuristics rely on a pre-defined graph with nodes evenly spaced around the table and edges connecting each adjacent node. We chose a density of 10 nodes per meter, this optimizes for speed of the simulation while enabling enough density for multiple options of shot selections.

For determining the base position based on the selected shot, we shortlist nodes around the table which have an angle between the cue ball and itself within the bounds:

$$[(shot\_angle - \pi) - \text{Offset}, (shot\_angle - \pi) + \text{Offset}]$$

We chose our offset to be $\frac{\pi}{2}$ after completing a small experiment to choose the most optimal angle. Finally, from the shortlist, we select the node closest to the cue ball as our "ideal" base position.

For moving around the table, we simply use Breadth First Search on the graph to get a trajectory for moving around the table.

While these heuristics are not optimal, their simplicity matches the difficulty of our problem and allows for quicker computation compared to more advanced methods of determining and moving to a base position, such as Reinforcement Learning and RRT-Connect.

F. Trajectory Planning for Striking Ball

We break up hitting the ball into a pre-shot pose and post-shot pose which the robot interpolates between. The pre-shot pose is solved for with an inverse kinematics problem that aligns the cue tip behind the ball at the correct shot angle and a predefined pitch angle of 20 degrees. While we move the base with the previous outlined method, we do not fully constrain the planar joint of the base in the problem allowing the base to move more, enabling greater shot flexibility. Additionally, we optimize the cost of the squared difference from nominal joint positions:

$$\sum_{q_i}(q_i - q_{nominal,i})$$

Given the pre-shot pose, the post-shot pose is solved for by advancing the cue tip forward along the desired strike direction by a fixed displacement while preserving the cue orientation. Specifically, the pre-shot inverse kinematics solution places the cue tip at a distance $(R + \text{gap})$ behind the cue ball along the shot normal and orients the cue at a fixed downward pitch of 20° toward the table. The post-shot pose is then defined by translating this configuration forward along the same direction so that the cue tip passes through the original contact point with the ball.

We then construct $N = 8$ intermediate ke yframes by linearly interpolating in task space between X_{pre} and X_{post}. For each interpolation parameter $\alpha_j \in [0, 1]$, the desired cue-tip pose is

$$X_j = (1 - \alpha_j)X_{pre} + \alpha_j X_{post},$$

where $\alpha_j = \frac{j}{N-1}$ for $j = 0, \ldots, N-1$. Each pose X_j is independently mapped back to joint space by solving the same inverse kinematics program used for the pre-shot pose, with the mobile base position locked and the cue orientation constrained to the fixed shot axis.

This yields a sequence of joint-space keyframes $\{q_0, \ldots, q_{N-1}\}$ that define the strike trajectory. These keyframes are executed sequentially using an inverse dynamics controller with short-duration interpolation between successive configurations. The resulting motion produces a fast, approximately straight-line cue motion through the ball, generating a high-impulse, dynamically consistent strike while maintaining precise directional control.

IV. Results

A. Evaluating Shot Precision

In order to gain insight on whether the controller is accurately following the desired trajectory needed to pocket balls. We measure our shot precision by calculating the angle between the desired angle given by our pool solver and the actual ball angle. We ran 10 random initializations and results shown in Table I.

TABLE I
Experimental Results: Angular Error Analysis

Trial No.	Angular Error (rad)
1	0.0071
2	0.0026
3	0.3699
4	0.0044
5	0.0015
6	0.0209
7	0.0033
8	0.0017
9	0.0044
10	0.0017
Mean	**0.0418**

With an average error of 0.0418 radians (2.39°), we can confidently say that our robot's trajectory is precise and most missed shots can be attributed to hard-to-follow trajectories

and inverse dynamics controller failure. This, however, is not the full story because the robot will not take shots when inverse kinematics fails.

B. Evaluating Multi-shot Sequences

To evaluate our goal of successfully playing multi-shot sequences, we count the number of attempted inverse kinematics solves and balls pocketed over 10 random initializations of four balls. Results shown in Table II.

TABLE II
SHOT PERFORMANCE STATISTICS OVER 10 GAMES

Attempts	Successful Shots	Consecutive Balls
4	4	4
5	4	4
7	4	3
9	3	2
5	4	2
3	2	2
8	1	1
4	2	2
4	4	4
5	2	2
Total: 54	**Total: 30**	**Total: 26**
Rate	*55.6%*	–

With 55.6 percent of attempted shots being successful, we leave more to be desired. Looking closer into each individual initialization, it is clear that this metric is skewed by instances when inverse kinematics failures occur. When it does not fail, PoolBot successfully pockets all four with minimal attempts. These realizations lead us to believe that with a more robust inverse kinematics problem, and a longer pool stick, balls could be pocketed more consistently.

CONCLUSIONS

PoolBot achieving 55% shot sucess rate and clearing an average of 2.6 balls per run demonstrates the feasibility of combining physics-aware optimization and mobile manipulation in a unified framework, that does not employ learning based methods, capable of multi-shot autonomous pool play. Our system improves on prior work through full-table mobility and a physics-aware shot planner. Although performance remains limited by simplified physics, heuristics, and robustness, our results indicate the viability of optimization-driven planning for contact-rich tasks. Future work includes (1) integration with a real robot platform, (2) incorporating more accurate friction modeling, (3) applying learning-based methods to optimize shot sequencing and base positioning, (4) improving cue stick trajectories via path planning algorithms (e.g. RRT) and a refined inverse kinematics formulation, and (5) adding perception to determine the game state.

ACKNOWLEDGMENT

An immense thank you to the 6.4210 course staff. We appreciate your unwavering help and dedication throughout this semester at lecture, office hours, and on Piazza. A special thank you to our project advisor Noah Fisher for his guidance and support. Additionally, another special thank you to our CI-M recitation leader Dave Larson for his insights into constructing a well written report.

COLLABORATION STATEMENT

TABLE III
TASK CONTRIBUTION RANKING (1ST = PRIMARY LEAD), *EVEN

Task	1st	2nd	3rd
CAD Modeling	Suraj R	Leo S	
Simulation Setup	Avinash N	Leo S	Suraj R
Collision Objects & Placement	Suraj R	Leo S	Avinash N
Motion Planning (Table)	Avinash N	Suraj R	Leo S
Pool Game Solver	Leo S	Suraj R	
Trajectory/IK (Striking)	Leo S	Avinash N	Suraj R
Friction Modeling	Avinash N		
Experiments	*Avinash N	*Leo S	*Suraj R
Paper Writing	Suraj R	*Leo S	*Avinash N
Video	Suraj R	*Leo S	*Avinash N

REFERENCES

[1] D. Greenspan, "Toward a Competitive Pool-Playing Robot," *IEEE Computer*. [Online]. Available: https://drdavepoolinfo.com/physics_articles/Greenspan_IEEE_08_article.pdf. Accessed: Oct. 30, 2025.

[2] Yale University, "Creating an Autonomous System to Play Pool," CPSC659 Project Report. [Online]. Available: https://cpsc659-bim.gitlab.io/f18/assets/reports/Pool.pdf. Accessed: Oct. 30, 2025.

[3] Stanford University, "Deep Cue Learning," CS229 Project Report. [Online]. Available: https://cs229.stanford.edu/proj2018/report/249.pdf. Accessed: Oct. 30, 2025.

[4] E. Feng, "6.4212 An Autonomous Pool-Playing Robot," YouTube Video. [Online]. Available: https://www.youtube.com/watch?v=578RsYdkPGs. Accessed: Oct. 30, 2025.